# Binary Search Trees

Scurtu Estera Daniela

May 30, 2016

Grupe 10106B
1st year
Computer and Information Technology(English)

# 1 Problem statement

A library for binary search trees (BST).

The library should provide operations for: creating an empty tree, inserting a node into a tree, deleting a node and at least two different traversal strategies; the traversal functions must accept a function which will be passed the current element.

# 2 Application Design

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left sub-tree of a node contains only nodes with keys less than the nodes key.
- The right sub-tree of a node contains only nodes with keys greater than the nodes key.
- The left and right sub-tree each must also be a binary search tree.
- There must be no duplicate nodes.

Binary Search Tree can be implemented as a linked data structure in which each node is an object with two pointer fields and an information field. The two pointer fields left and right point to the nodes corresponding to the left child, right child. NULL in any pointer field signifies that there exists no corresponding child.

## 2.1 Inputs

Input data is read from a file named test.in.txt which has the following structure:

- the first line contains the number of nodes that are initially in the tree
- the second line contain the number of tests to be made
- the next nrTeste lines, has each two values, one for the nodes to be inserted in the tree and one for the nodes to be deleted from the tree.

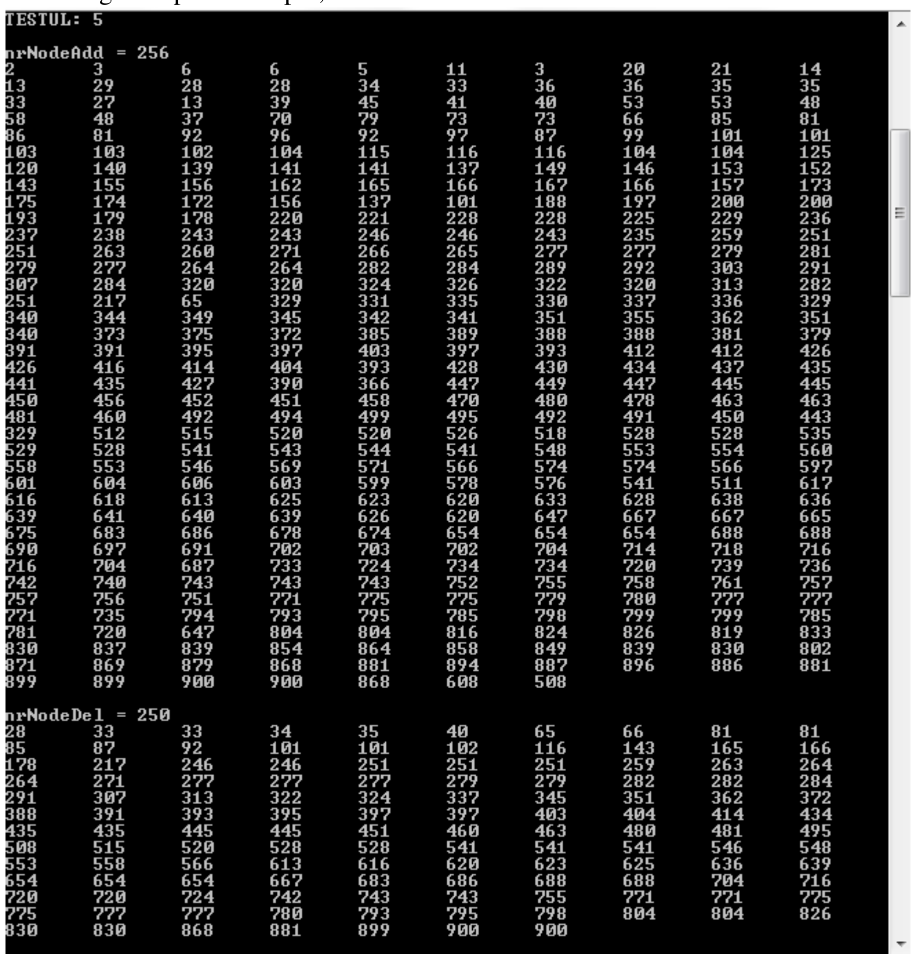
Here is an example of inputs for this algorithm:
150
10
270 300
256 250
255 237
153 268
240 169
250 130
122 202
130 110

70 123
125 213

## 2.2 Outputs

First output is the number of nodes of the tree followed by a traversal of the tree, breadth first tree traversal. The rest output consists in nrTeste tests which include inserting nrNodeAdd in the tree and deleting nrNodeDel from the tree.

This image is a part of output, the test number 5.



```
TESTUL: 5

nrNodeAdd = 256
2       3       6       6       5       11      3       20      21      14
13      29      28      28      34      33      36      36      35      35
33      27      13      39      45      41      40      53      53      48
58      48      37      70      79      73      73      66      85      81
86      81      92      96      92      97      87      99      101     101
103     103     102     104     115     116     116     104     104     125
120     140     139     141     141     137     149     146     153     152
143     155     156     162     165     166     167     166     157     173
175     174     172     156     137     101     188     197     200     200
193     179     178     220     221     228     228     225     229     236
237     238     243     243     246     246     243     235     259     251
251     263     260     271     266     265     277     277     279     281
279     277     264     264     282     284     289     292     303     291
307     284     320     320     324     326     322     320     313     282
251     217     65      329     331     335     330     337     336     329
340     344     349     345     342     341     351     355     362     351
340     373     375     372     385     389     388     388     381     379
391     391     395     397     403     397     393     412     412     426
426     416     414     404     393     428     430     434     437     435
441     435     427     390     366     447     449     447     445     445
450     456     452     451     458     470     480     478     463     463
481     460     492     494     499     495     492     491     450     443
329     512     515     520     520     526     518     528     528     535
529     528     541     543     544     541     548     553     554     560
558     553     546     569     571     566     574     574     566     597
601     604     606     603     599     578     576     541     511     617
616     618     613     625     623     620     633     628     638     636
639     641     640     639     626     620     647     667     667     665
675     683     686     678     674     654     654     654     688     688
690     697     691     702     703     702     704     714     718     716
716     704     687     733     724     734     734     720     739     736
742     740     743     743     743     752     755     758     761     757
757     756     751     771     775     775     779     780     777     777
771     735     794     793     795     785     798     799     799     785
781     720     647     804     804     816     824     826     819     833
830     837     839     854     864     858     849     839     830     802
871     869     879     868     881     894     887     896     886     881
899     899     900     900     868     608     508

nrNodeDel = 250
28      33      33      34      35      40      65      66      81      81
85      87      92      101     101     102     116     143     165     166
178     217     246     246     251     251     251     259     263     264
264     271     277     277     277     279     279     282     282     284
291     307     313     322     324     337     345     351     362     372
388     391     393     395     397     397     403     404     414     434
435     435     445     445     451     460     463     480     481     495
508     515     520     528     528     541     541     541     546     548
553     558     566     613     616     620     623     625     636     639
654     654     654     667     683     686     688     688     704     716
720     720     724     742     743     743     755     771     771     775
775     777     777     780     793     795     798     804     804     826
830     830     868     881     899     900     900
```

If we were to run again the program, the numbers would be different because the program would generate different numbers for each run.

## 2.3 Operations on Binary Search Trees

1. Create an element

   This operation can be seen in the procedure "getNode" which have a parameter of type int, that represent the value we want to store in the created node. Firstly I allocate memory for one element of type "bstNode" using the "malloc" function, next step is to store the value key in the data field. The last step to be taken is to initialize the right and left fields of the node to NULL.

   I will use this function in the procedure "insert_bst" to create the node to be inserted in the tree.

2. Insertion of a node in the tree

   For this operation I used a function named "insert_bst" with two parameters one of type bstNode which is the adress of the root, and an integer which represents the value to be inserted. Insertion begins as a search would begin; if the key is not equal to that of the root, we search the left or right subtrees as before. Eventually, we will reach an external node and add the new key-value pair as its right or left child, depending on the node's key. In other words, we examine the root and recursively insert the new node to the left subtree if its key is less than that of the root, or the right subtree if its key is greater than or equal to the root

3. Deletion of a node of the tree

   This operation is described in the procedure "deleteNode" with two parameters, one that represent the adress of the root and another one that takes the value to be deleted. Deletion also begins as a search, we try to find where the node that has the value we want to delete is, if the key is lesser than the root(the actual node) value we use recursion and call the function to move in the left sub-tree and take the steps from the beginning, else if is greater we move in the right sub-tree, else we find the wanted value and from here three cases can be distinguished:

   - Case 1: the node has no child - this case is the simple one because the wanted node already points to NULL so all we have to do is to deallocate the node using the function free and make it NULL.
   - Case 2: the node has one child - here there can be two cases, the node can have either the left child or the right child. If the node has only a left child, we put all the left sub-tree int the place of the node to be deleted. I made this using an auxiliary variable named temp which will copy the address of the root, and make the root point to its left child, while the auxiliary variable will be deallocated. The case for the right child is similar the only difference is where the root node will point.

- Case 3: the node has two children - to solve this case I will use a function ”findMin” which goes to the most left child of the tree, which is the minimum value and return its address. I will use the auxiliary variable again, this time it will store the address of the minimum value in the right sub-tree, we copy this value to the node we want to delete. Now there will be two nodes that store the same value, because we don’t know in which one of the cases we are we will call the function with the parameters root-¿right and root-¿data, which is the minimum value in the right sub-tree.

4. Traversal of a tree.

   There are two known types of traversal for a Binary Search Tree:

   (a) Depth First Search which can be
       - Pre-order Traversal
       - In-order Traversal
       - Post-order Traversal

   (b) Breadth First Search - to implement this traversal we will use three functions which take as a parameter the address of the root: ”height” which compute the height of the tree the number of nodes along the longest path from the root node down to the farthest leaf node, ”printGivenLevel” which print the node of a given level, ”level” is a parameter of the procedure and the last function is printLevelOrder print all the nodes in level order.

5. Search for a given value in the tree

   The procedure for this task is named ”search_node” and takes two parameters, one of type bstNode, which is the adress of the root, and one integer which is the value we want to find. First we compare the given value with the data stored in the node, if it is greater than the node’s data we move the search in the right subtree, else we are looking for the value in the left sub-tree. Once we find the value we retuen its adress.

6. Add elements in the tree

   The procedure used for this operation is create_bst. This function use the function radomize which generate random numbers that are inserted in the tree using the insert_bst function. To prevent the insertion of the same number I used the goto function which force the algorithm to reread a number if it exists in the tree.

7. Delete elements from the tree

   The procedure used for this operation is nodeDelete. This function generate random numbers which are delete from the tree using the deleteNode function. Because I can not know which are the current numbers of the tree and also the numbers generated randomly I check if the data is in the tree first, if it is it will be deleted, it it is not the program will generate another number to be deleted.

## 3 Pseudocode

- The procedure insert_bst

procedure insert_bst (var root :struct bstNode;
    var key : integer) : struct bstNode
begin
    **if** root = NULL **then**
        root := getNode(var root ; var key)
    end
    else if $root \rightarrow data < key$ then
        $root \rightarrow left := insert\_node(root \rightarrow left, key)$
    end
    else if $root \rightarrow data > key$ then
        $root \rightarrow right := insert\_node(root \rightarrow right, key)$
    end
    return root
end insert_node

- The procedure deleteNode

procedure deleteNode (var root :struct bstNode;
    var key : integer) : struct bstNode
begin
    var temp : bstNode
    if $root = NULL$ then
        return root;
    end
    if $root \rightarrow data < key$ then
        $root \rightarrow left := deleteNode(root \rightarrow left, key)$
    end
    else if $root \rightarrow data > key$ then
        $root \rightarrow right := deleteNode(root \rightarrow right, key)$
    end
    else if ! $root \rightarrow left$ and ! $root \rightarrow right$ then
        free root
        root := NULL
    end
    else if $root \rightarrow left = NULL$ then

$temp$ := $root$
$root$ := $root \rightarrow right$
free temp
end
else if $root \rightarrow left$= $NULL$ then
$temp$ :=$root$
$root$ := $root \rightarrow right$
free temp
end
else if $root \rightarrow left$ and $root \rightarrow right$ then
$temp$:= $findMin(root \rightarrow right)$
$root \rightarrow data$ := $temp \rightarrow data$
$root \rightarrow right$ := $deleteNode(root \rightarrow right, root \rightarrow data)$
end
end deleteNode

- The procedure search_node

procedure search_node (var root :struct bstNode;
var key : integer) : struct bstNode
begin
**if** root = NULL **then**
return false
end
if $root \rightarrow data = key$ then
return true
end
else if $root \rightarrow data < key$ then
return $search\_node(root \rightarrow left, key)$
end
else if $root \rightarrow data > key$ then
return $search\_node(root \rightarrow roght, key)$
end
end insert_node

# 4 Conclusions

Only using linked lists may not be enough in some applications, for example if we want a description of a product, we would need an hierarchical description of its components. Hierarchical organisation of data is used in various fields and we can say that every physical entity can be represented as trees. The binary search tree is a different way of structuring data in hierarchical way so that it can still be binary searched (or a very similar procedure can be used), but it's easier to add and remove elements. The implementation based on dynamic programming makes the code clear and easily understandable and reduce the complexity of the algorithm.

The algorithm I made presents the most important operations that can be made on binary search trees, operations like insertion, deletion, traversals and binary search. The most challenging part of the project was to write the procedure for deletion because it is very complex and it took a lot of time to realise which was my mistake.

# 5 References

# References


[1] Sara Baase, Computer Algorithms, Introduction to Design and Analysis *Edison-Wesley Publishing Company*. Edison Wesley, 2nd Edition,

[2] Ellis Horowitz, Fundamentals of Programming Languages, *Computer Science Press*. Computer Science Press, 2nd Edition,

[3] site url http://www.geeksforgeeks.org/category/binary-search-tree/,.


# 6 Source Code

Here are all the function I described above implemented in C

```
#ifndef BST_H_H_INCLUDED
#define BST_H_H_INCLUDED
#include "bst_c.c"

struct bstNode *getNode(int key);

struct bstNode *insert_bst(struct bstNode *root,int key);

bool search_node(struct bstNode *root, int key);

struct bstNode *create_bst(struct bstNode *root, int nrNode);

int bstSrd(struct bstNode *root);

int bstRsd(struct bstNode *root);

int bstSdr( struct bstNode *root);

struct bstNode *findMin(struct bstNode *root);

struct bstNode *deleteNode(struct bstNode *root, int key);

int printGivenLevel(struct bstNode* root, int level);
```

```
int height(struct bstNode* root);

void printLevelOrder(struct bstNode* root);

void freeTree(struct bstNode *root);

void nodeDelete(struct bstNode *root, int n);
#endif // BST_H_H_INCLUDED


#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include "bst_h.h"
int randomize();
struct bstNode{

    int data;
    struct bstNode *left;
    struct bstNode *right;
};
struct bstNode *getNode(int key){
    struct bstNode *node = (struct bstNode *) malloc( sizeof ( struct bstNode));
    node->data = key;
    node->left = node->right = NULL;
    return node;
}

struct bstNode *insert_bst(struct bstNode *root, int key){
    if(root == NULL){
        root = getNode(key);
    }
    else if( key < root->data){
        root->left = insert_bst( root->left, key);
    }
    else{
        root->right = insert_bst( root->right, key);
    }
    return root;
}

bool search_node(struct bstNode *root, int key){
    if(root == NULL){
        return false;
    }
```

```
    else if( root->data == key){
        return true;
    }
    else if(key < root->data){
        return search_node(root->left , key);
    }
    else{
        return search_node(root->right , key);
    }
}

struct bstNode *create_bst(struct bstNode *root , int nrNode){
    int i , data;
    for( i = 0; i < nrNode; i++){
pas0:   data = rand() % 900 + 1;
        if(search_node(root ,data)){
            goto pas0;
        }
        else{
            root = insert_bst(root , data);
        }
    }
}

int bstSrd(struct bstNode *root){

    if( root == NULL){
            return 0;
    }
    bstSrd(root->left);
    printf("%d\t", root->data);
    bstSrd(root->right);

}

int bstRsd(struct bstNode *root){
    if(root == NULL){
            return 0;
    }
    printf("%d\t", root->data);
    bstRsd(root->left);
    bstRsd(root->right);
}

int bstSdr( struct bstNode *root){
    if(root == NULL){
```

```
            return 0;
    }
        bstSdr(root->left);
        bstSdr(root->right);
        printf("%d\t",root->data);

}

int height(struct bstNode* root)
{
    if (root==NULL)
        return 0;
    else
    {
/** compute the height of each subtree */
        int leftHeight = height(root->left);
        int rightHeight = height(root->right);

/** use the larger one */
        if (leftHeight > rightHeight){
            return(leftHeight+1);
        }
        else{
                return (rightHeight+1);
        }
    }
}

int printGivenLevel(struct bstNode* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d\t", root->data);
    else if (level > 1)
    {
        printGivenLevel(root->left, level-1);
        printGivenLevel(root->right, level-1);

    }
}

void printLevelOrder(struct bstNode* root)
{
    int h = height(root);
    int i;
```

```
    for (i=1; i<=h; i++){
        printGivenLevel(root, i);

    }
}

struct bstNode *findMin(struct bstNode *root){
    if(root == NULL){
        printf("the tree is empty");
        return 0;
    }
    else if( root->left == NULL){
        return root;
    }

    return findMin(root->left);
}

struct bstNode *deleteNode(struct bstNode *root,int key) {
    struct bstNode * temp;
    if (root == NULL) {
        return root;
    }
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    }
    else {
        if(root->left == NULL && root->right == NULL){
            free(root);
            root = NULL;
        }
        else if (root->left == NULL) {
                temp = root;
                root = root->right;
                free(temp);
        }
        else if (root->right == NULL) {
                    temp = root;
                    root = root->left;
                    free(temp);
            }
        else if (root->left != NULL  && root->right != NULL) {
                temp = findMin(root->right);
                root->data = temp->data;
                root->right = deleteNode(root->right, root->data);
            }
```

```c
        }

    return root;
}

void freeTree(struct bstNode *root) {
    if (root == NULL) {
        return;
    }
    freeTree(root->left);
    freeTree(root->right);
    free(root);
}

void nodeDelete(struct bstNode *root, int n){
    int i;
    int data;
    for(i = 0; i < n; i++ ){
pas1:   data = rand() % 900;
        if(search_node(root,data)){
            root = deleteNode(root, data);
        }
        else{
            goto pas1;
        }
    }
}

#include "bst_h.h"
int main()
{
    srand(time(NULL));
    FILE *test;
    test = fopen("test.in.txt","r");

    int nrTeste;
    int nrNode;
    int nrNodeAdd;
    int nrNodeDel;

    struct bstNode *root = NULL;
    root = insert_bst(root, 500);

    fscanf(test,"%d",&nrNode);
    fscanf(test,"%d",&nrTeste);
    printf("Numarul de elemente din arbore este %d\n",nrNode);
```

```
    create_bst(root, nrNode);
    printLevelOrder(root);
    printf("\n\n\n");
    while(nrTeste != 0){
        printf("\n\nTESTUL: %d\n ",nrTeste);
        fscanf(test,"%d %d",&nrNodeAdd, &nrNodeDel);
        printf("\nnrNodeAdd = %d\n", nrNodeAdd);
        create_bst(root, nrNodeAdd);
        bstSdr(root);
        printf("\n\nnrNodeDel = %d\n",nrNodeDel);
        nodeDelete(root,nrNodeDel);
        bstSrd(root);

        nrTeste--;

    }
    freeTree(root);
    fclose(test);
}
```

Here is the code in Python3.5 for the algorithm. I must say that this code does not work entirely, it is a problem when trying to delete more numbers. I tried to fix this problem but I could not do that.

```
class Node:
    def __init__(root, val):
        root.right = None
        root.left = None
        root.data = val

def insert(root, data):
    if root.data:
        if data < root.data:
            if root.left is None:
                root.left = Node(data)
            else:
               insert(root.left,data)
        elif data > root.data:
            if root.right is None:
                root.right = Node(data)
            else:
                insert(root.right, data)
    else:
        root.data = data

def SRD(root):
    if root is None:
```

```
        return
    SRD(root.left)
    print (root.data)
    SRD(root.right)

def RSD(root):
    if root is None:
        return
    print (root.data)
    RSD(root.left)
    RSD(root.right)

def bin_search(root, data):
    if root is None:
        return 0
    elif root.data is data:
        return 1
    elif data < root.data:
        return bin_search(root.left,data)
    else:
        return bin_search(root.right,data)

def find_min(root):
    if root is None:
        print ("the tree is empty")
        return
    elif root.left is None:
        return root
    return find_min(root.left)

def delete_node(root, data):
    if root is None:
        return root
    if data < root.data:
        root.left = delete_node(root.left, data)
    elif data > root.data:
        root.right = delete_node(root.right, data)
    else:
        if root.left is None and root.right is None:
            del root
            root = None
        if root.left is None :
            temp = root.right
            root = None
            return temp
        elif root.right is None :
```

```
                temp = root.left
                root = None
                return temp
            temp = find_min(root.right)
            root.data = temp.data
            root.right = delete_node(root.right , temp.data)
        return root

def create_bst(root,n):
    for i in range (1,n):
        a = randint(0,300)
        insert(root,a)

def delete_bst(root,n):
    for i in range (1,n):
        a = randint(0,300)
        delete_node(root,a)


from random import randint
root = Node(110)
create_bst(root,150)
SRD(root)
delete_bst(root,50)
RSD(root)
```

# 7 Experiments and results

To check the correctness of the algorithm I made 10 tests, each consisting in inserting a number of nrNodeAdd nodes in the tree and deleting a number nrNodeDel from the tree, the nrNodeAdd and nrNodeDel are two variables read from a file. The results were displayed using three traversal functions: level order traversal, to display the initial tree, in-order traversal, to print the tree after the insertion and post-order traversal for printing the remaining nodes after deletion.